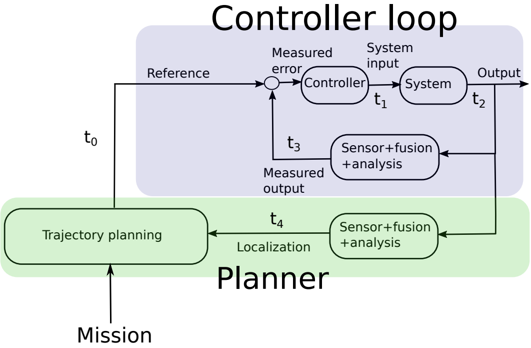
Extended Reliable Robust Motion Planners

Adina M. Panchea[1]

joint work with **Alexandre Chapoutot**[2], **David Filliat**[2]
[1]LIX, Ecole Polytechnique, Palaiseau, France
[2]U2IS, ENSTA ParisTech, Palaiseau, France

March 28, 2017

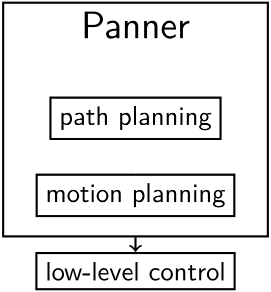
Autonomous vehicle



Goal of the project:

- understand main pieces of the system
- validate their behaviour
- validate the behaviour of the overall system.

A hierarchical control



- **Path planning** generates a set of way points (does not take into account the dynamics of the vehicle) from a map (totally or partially) known, take into account obstacles (static)
- **Motion planning** generates a set of trajectories feasible for the dynamics considered and take into account obstacles (static and/or dynamic)
- **Low-level controller** tries to follow the (discretized) trajectory w.r.t. the dynamic of the vehicle

This talk

We focus on sampling-based **motion planning algorithms** :

- Rapidly-exploring Random Trees (**RRT**s) and
- Optimal Rapidly-exploring Random Trees (**RRT***).

Take into account

- A model of the vehicle,
- A model of a map with obstacles (static),
- Uncertain information on the position/orientation, etc. - bounded within *interval vectors* or *boxes*.

This talk

Propose new methods to plan guaranteed to be safe paths:

- improved BoxRRT - **rciBoxRRT**,
- improved BoxRRT - **csiBoxRRT**,
- new algorithm based RRT* - **t**(towards)**BoxRRT***.

The BoxRRT is based on

Reliable robust path planning. Romain Pepy, Michel Kieffer, Eric Walter. Journal Applied Mathematical Computing. 2009.

Box-RRT Algorithm

Goal

- "quickly" find a path going from an initial configuration $\mathbf{s}_i$ to a final configuration $\mathbf{s}_f$
- while avoiding obstacles $\mathbf{s}_o$
- and taking into account bounded uncertainties.

Main ingredients

- model of the vehicle
- based on RRT Algorithm
- combined with interval analysis tools (*e.g.*, guaranteed numerical integration)
- applied with
  1. a random (**rciBoxRRT**) and
  2. a designed control input (**sciBoxRRT**)

First improvement : use of modern and new tools for the guaranteed numerical integration

Implement these algorithms with **DynIBEX**.

DynIBEX in few words

A library combining of **Constraint Satisfaction Problems solver** (IBEX[1]) with **validated numerical integration methods** à la Runge-Kutta.

Three temporal constraints have been used:

- stay in state-space;
- has crossed (in negative form) obstacles;
- one in target;

and one contractor on tube has been used

- get tight(t) to get final state.

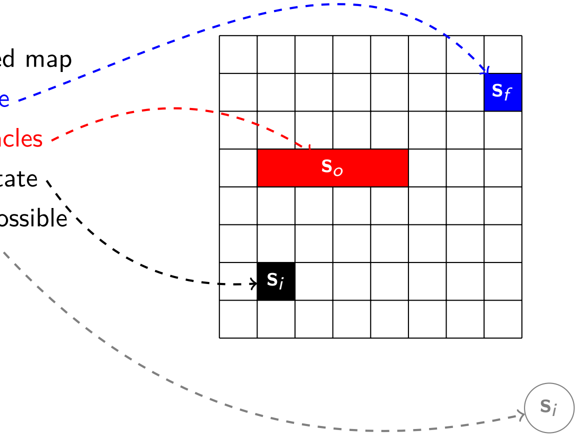
So these algorithms can be quickly implemented in DynIBEX.

[1] Gilles Chabert (EMN) et al. `http://www.ibex-lib.org`

How does rciBoxRRT−sciBoxRRT work?

Initialization

- A discretized map
- A final state
- Some obstacles
- An initial state
- A tree of possible trajectories



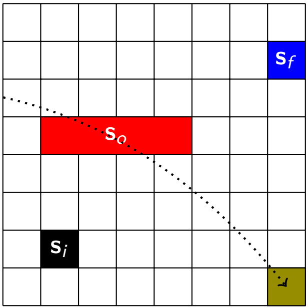
How does rciBoxRRT−sciBoxRRT work?

Iteration 1

The building of the tree follows 3 main steps in a loop

1. Pick s_{random}
2. Find s_{nearest}
3. Compute s_{new}

until $\mathbf{s}_f$ is reached



Choose randomly a free state of the map

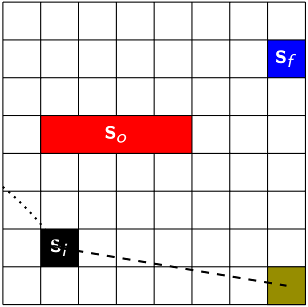
How does rciBoxRRT−sciBoxRRT work?

Iteration 1

The building of the tree follows 3 main steps in a loop

1. Pick s_{random}
2. Find s_{nearest}
3. Compute s_{new}

until $\mathbf{s}_f$ is reached



Find the nearest node in the tree following a Hausdorff distance between two boxes

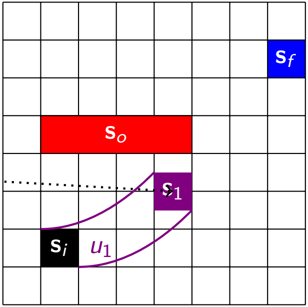
How does rciBoxRRT−sciBoxRRT work?

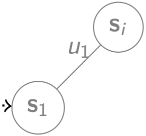
Iteration 1

The building of the tree follows 3 main steps in a loop

1. Pick s_{random}
2. Find s_{nearest}
3. Compute s_{new}

until $\mathbf{s}_f$ is reached





Predict the next state from a **random** or **designed** control input u

if no collision detected add ($\mathbf{s}_{\text{nearest}}$, u, $\mathbf{s}_{\text{new}}$) in the tree

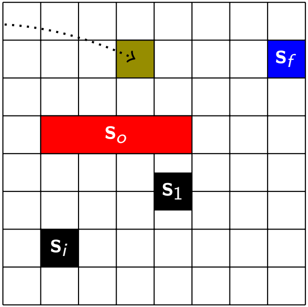
How does rciBoxRRT−sciBoxRRT work?

Iteration 2

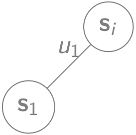
The building of the tree follows 3 main steps in a loop

1. Pick s_{random}
2. Find s_{nearest}
3. Compute s_{new}

until $\mathbf{s}_f$ is reached



Note that the choice of $\mathbf{s}_{\text{random}}$ is biased to increase probability to be closer to $\mathbf{s}_f$ (*Random box BiasGoal procedure*)



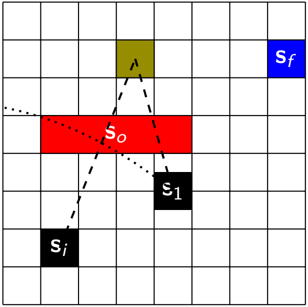
How does rciBoxRRT−sciBoxRRT work?

Iteration 2

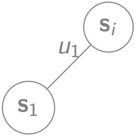
The building of the tree follows 3 main steps in a loop

1. Pick s_{random}
2. Find s_{nearest}
3. Compute s_{new}

until $\mathbf{s}_f$ is reached



Computing distances has a linear complexity w.r.t. the number of nodes in the tree



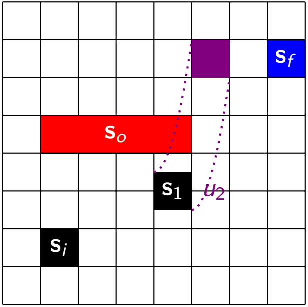
How does rciBoxRRT−sciBoxRRT work?

Iteration 2

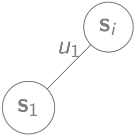
The building of the tree follows 3 main steps in a loop

1. Pick s_{random}
2. Find s_{nearest}
3. Compute s_{new}

until $\mathbf{s}_f$ is reached



The absence of collision is detected when the **tube** does not intersect an obstacle



tBoxRRT* Algorithm

RRT* Algorithm

- "quickly" finds a "low cost" path going from an initial configuration $\mathbf{s}_i$ to a final configuration $\mathbf{s}_f$
- while avoiding obstacles $\mathbf{s}_o$

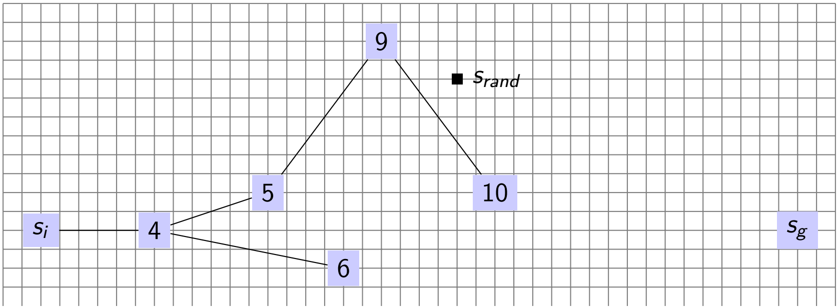
Based on :

Sampling-based algorithms for optimal motion planning. S. Karaman and E. Frazzoli. The international journal of robotics research. 2011.

tBoxRRT* Algorithm

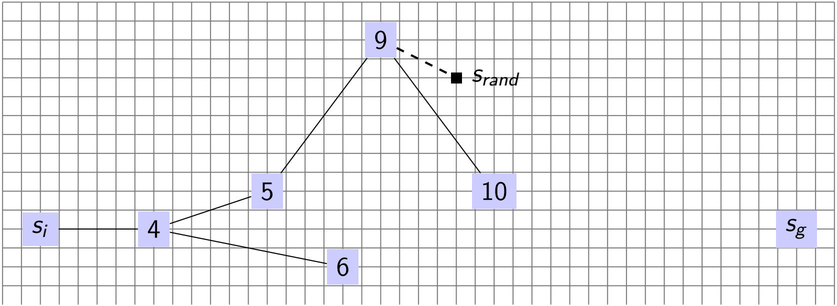
- based on RRT* Algorithm : finds a "low distance" path going from an initial configuration $[\mathbf{s}_i]$ to a final configuration $[\mathbf{s}_f]$,
- combined with interval analysis tools (*e.g.*, guaranteed numerical integration)
- while avoiding obstacles $\mathcal{S}_{obs}$

How does tBoxRRT* work?



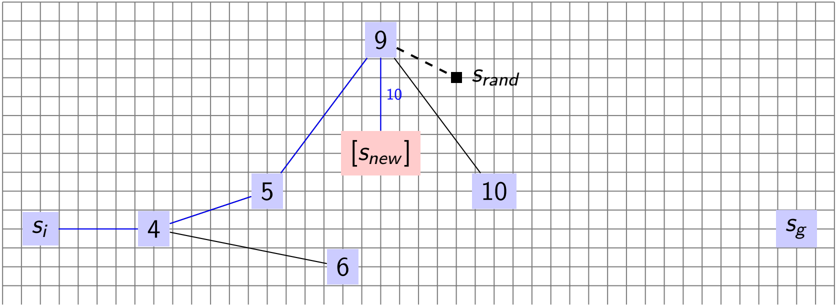
- $[s_{rand}] \leftarrow$ random-box-GoalBias;
- Values inside the vertices : distance (e.g. the Hausdorff distance between two boxes) from the initial state to that vertex.

How does tBoxRRT* work?



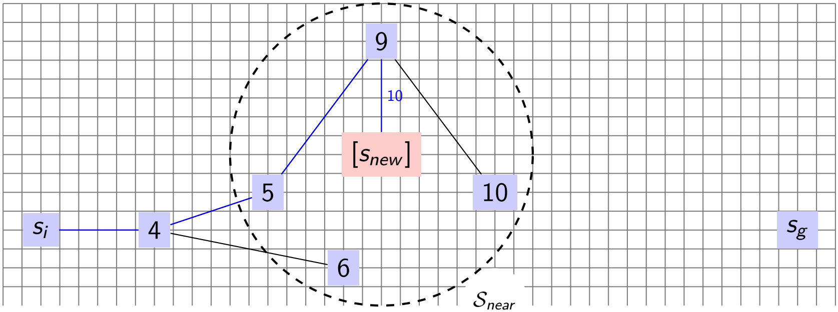
- $[s_{nearest}] \leftarrow$ nearest-neighbor$(G, [s_{rand}])$;
- Nearest-neighbor procedure uses the Hausdorff distance between two boxes metric.

How does tBoxRRT* work?

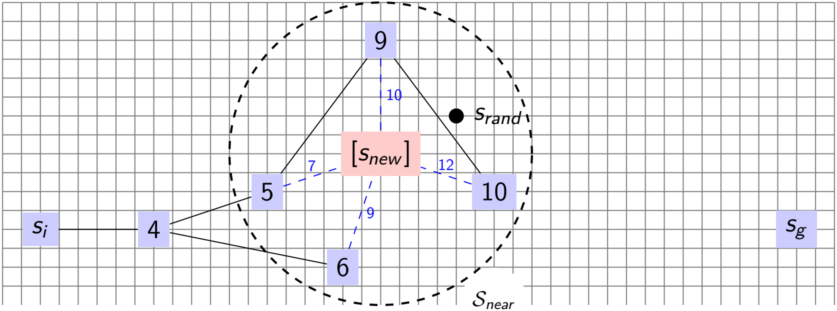


- $([s_{new}], u) \leftarrow$ steer $([s_{nearest}], [s_{rand}])$
- 1. u is computed using a desired objective.

How does tBoxRRT* work?

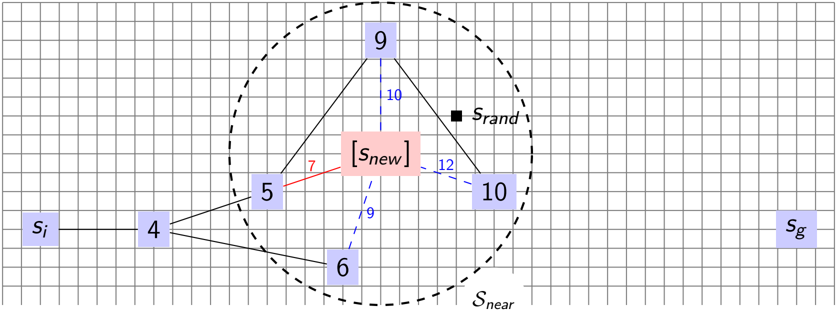


- $\mathcal{S}_{near} \leftarrow$ near $(G, [s_{new}])$
- Near procedure : uses **k-nearest neighbors algorithm** (all vertices within the area of a ball of radius $r(n) = \gamma log(n)$ with $\gamma = 2\epsilon$ (ϵ : Euler's number; n : number of vertex in the tree at an iteration))



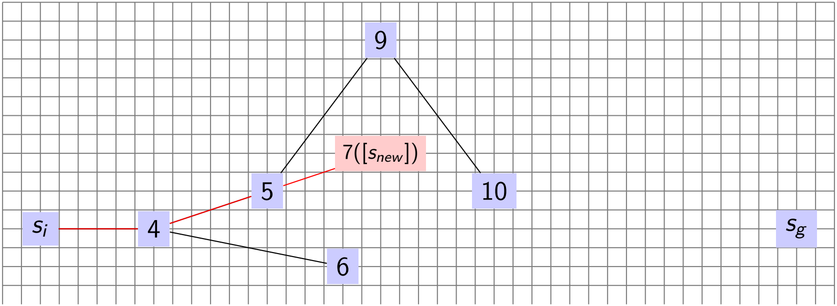
- $[s_{min}] \leftarrow$ ChooseParent $(\mathcal{S}_{near}, [s_{nearest}], [s_{new}])$

How does tBoxRRT* work?



- $[s_{min}] \leftarrow$ ChooseParent $(\mathcal{S}_{near}, [s_{nearest}], [s_{new}])$

How does tBoxRRT* work?



- $G \leftarrow \text{rewire}(G, [s_{min}], [s_{new}])$

Until

- *max iteration number is reached* or
- *a solution is found (e.g* $[s_{new}] \neq \emptyset$, $[s_{new}] \subset Int([s_{goal}])$*)*

tBoxRRT* Algorithm

input : $[s_{init}]$, $[s_{goal}]$, K;
output : $G = (V, E)$;
$G.init([s_{init}])$;
i $\leftarrow$ 0 ;
repeat
  $[s_{rand}] \leftarrow$ random-box(i)
  $[s_{nearest}] \leftarrow$ nearest-neighbor($G, [s_{rand}]$)
  $([s_{new}], u) \leftarrow$ steer($[s_{nearest}], [s_{rand}]$)
  if collision-free-path($[s_{new}]$) **then**
    $\mathcal{S}_{near} \leftarrow$ near($G, [s_{new}], V$)
    $[s_{min}] \leftarrow$ ChooseParent($\mathcal{S}_{near}, [s_{nearest}], [s_{new}]$)
    $G \leftarrow$ rewire($G, [s_{min}], [s_{new}]$)
  end if
until $(i++<K)$ or $([s_{new}] \neq \emptyset, [s_{new}] \subset Int([s_{goal}]))$
return G

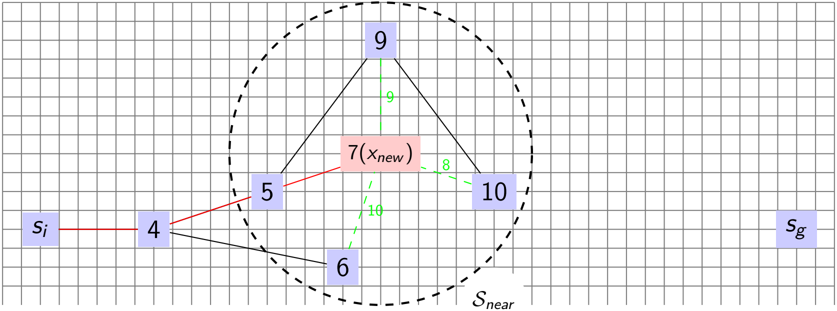
Algorithm 1: BoxRRT* motion planning algorithm

BoxRRT* Algorithm - **Future work**

input : $[s_{init}]$, $[s_{goal}]$, K;
output : $G = (V, E)$;
$G.init([s_{init}])$;
i $\leftarrow$ 0 ;
repeat
  $[s_{rand}] \leftarrow$ random-box(i)
  $[s_{nearest}] \leftarrow$ nearest-neighbor($G, [s_{rand}]$)
  $([s_{new}], u) \leftarrow$ steer($[s_{nearest}], [s_{rand}]$)
  if collision-free-path($[s_{new}]$) **then**
    $\mathcal{S}_{near} \leftarrow$ near($G, [s_{new}], V$)
    $[s_{min}] \leftarrow$ ChooseParent($\mathcal{S}_{near}, [s_{nearest}], [s_{new}]$)
    $[s_{min_k}] \leftarrow$ ChooseChildren($\mathcal{S}_{near} \setminus \{[s_{min}]\}, [s_{new}], [s_{nearest}]$)
    $G \leftarrow$ rewire($G, [s_{min}], [s_{new}], [s_{min_k}]$)
  end if
until $(i{+}{+} < K)$
return G

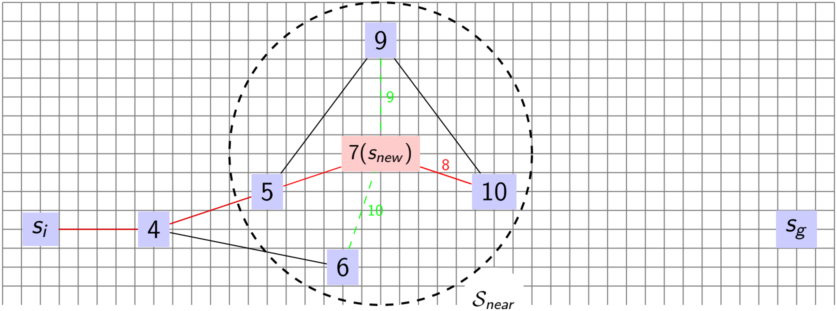
Algorithm 2: BoxRRT* motion planning algorithm

How does ChooseChildren procedure work?



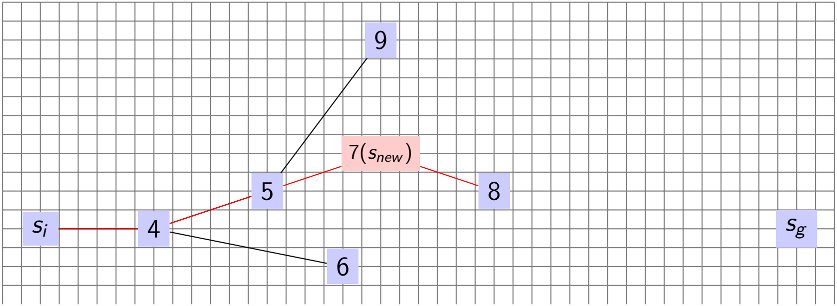
- $[s_{min_k}] \leftarrow \text{ChooseChildren}(\mathcal{S}_{near} \setminus \{[s_{min}]\}, [s_{new}], [s_{nearest}])$

How does ChooseChildren procedure work?



- $[s_{min_k}] \leftarrow$ ChooseChildren$(\mathcal{S}_{near} \setminus \{[s_{min}]\}, [s_{new}], [s_{nearest}])$

How does ChooseChildren procedure work?



- $G \leftarrow \text{rewire}(G, [s_{min}], [s_{new}], [s_{min_k}])$

Future work :
Until the max iteration number is reached.
Apply the A$*$ method and find the shortest path? ....

Cinematic of a Mobile Robot in 2D

We consider a simple car model

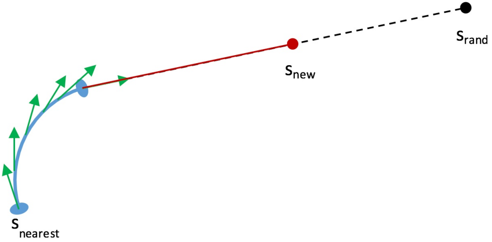
$$\dot{x} = v\cos(\theta)$$
$$\dot{y} = v\sin(\theta)$$
$$\dot{\theta} = \frac{v}{L}\tan(\delta)$$

with constraints

- $v \in [-1, 1]$ - longitudinal speed and
- $\delta \in [-\pi/2, \pi/2]$ - steering angle.
- $L = 1.5[m]$ - distance between the front and back axes of the car.

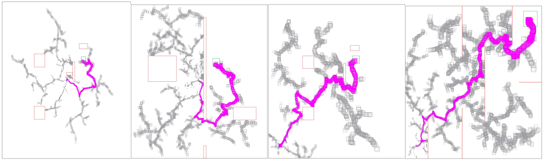
Control input for the simple car model

- **rciBoxRRT** : control input randomly chosen in the admissible set.
- **sciBoxRRT** and **tBoxRRT***: control input designed in two steps:

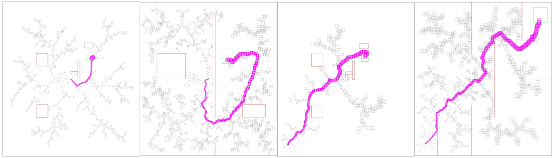


Results: 4 environments

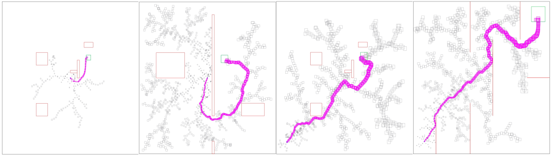
- ***rciBoxRRT***(2200t.v.:28[s]; 5880 t.v.:103[s]; 3416t.v.:51 [s]; 7802t.v.:141[s]).



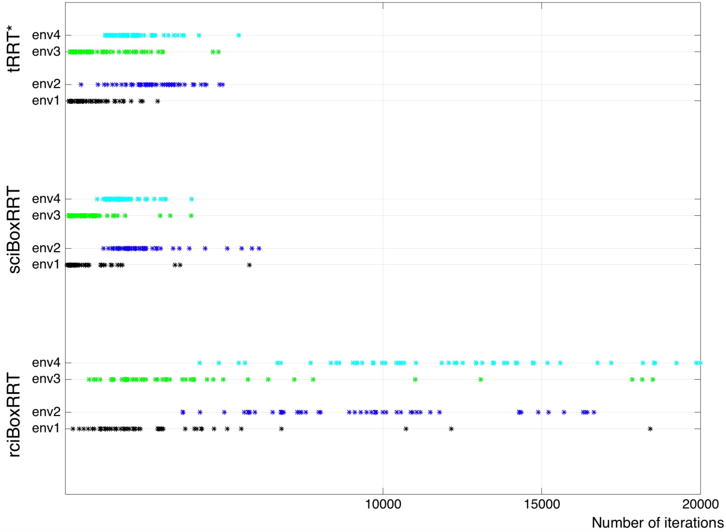
- ***sciBoxRRT***(570 t.v.:11 [s]; 1149 t.v.:32 [s]; 278 t.v.:5[s]; 978 t.v.:26[s]).



- ***tBoxRRT****(156 t.v.:3 [s]; 1088 t.v.:38 [s]; 786 t.v.:20 [s]; 963 t.v.:28[s]).

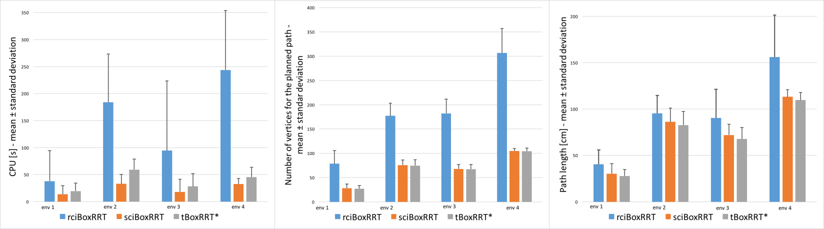


Results: rciBoxRRT, sciBoxRRT and tBoxRRT*



Results: rciBoxRRT, sciBoxRRT and tBoxRRT*

- Computational time (s) required by the three proposed algorithms for convergence,
- Number of vertices for the planned path obtained by the three proposed algorithms,
- Planned path length (cm) obtained by the three proposed algorithms.



Conclusion

- Shown three motion planner algorithms in the control hierarchy: rciBoxRRT, sciBoxRRT and tBoxRRT$*$ (Submitted to CDC17)
- Presented a small example of autonomous vehicle

Future work

- Propose the BoxRRT$*$ (maybe use the A$*$ method for the shortest path search)
- BiBoxRRT$*$: based on RRT Algorithm with two trees : one growing from $\mathbf{s}_i$ and the other one from $\mathbf{s}_f$ until they intersect ?
- Combine Box-RRT with low-level controller (PID)

Acknowledgment

Thanks to

- **Olivier Mullier**, **Julien Alexandre dit Sandretto**, (U2IS, ENSTA ParisTech, Palaiseau, France)
- **Eric Goubault** and **Benjamin Martin**, (LIX, Ecole Polytechnique, Palaiseau, France)

for productive and useful discussions.

Thank you !